



Programación Python para BigData

Lección 7: Apache Spark con PySpark
[1/2]

ÍNDICE

Lección 7: Apache Spark con PySpark [1/2]	3
Presentación y objetivos	3
1. Spark	4
2. Instalación Apache Spark	8
3. Spark SQL y Dataframe	11
4. Spark RDD	18
5. Puntos claves	28

Lección 7: Apache Spark con PySpark [1/2]

PRESENTACIÓN Y OBJETIVOS

En esta lección aprenderemos a trabajar con Spark usando una librería de Python como es PySpark, para ello usaremos una imagen de Docker y veremos los distintos usos con los que podemos trabajar.



Objetivos

- | Conocer qué es Spark y sus usos en **Big Data**.
- | Usar la imagen de pyspark-notebook para trabajar con este entorno.
- | Conocer los usos de Pyspark librería de Python en **Big data**.

1. SPARK

Introducción a Spark

Apache Spark es un Framework de programación Open Source para procesamiento de datos distribuidos diseñado para ser rápido. Es un sistema de computación que se basa en Hadoop Map Reduce y que, principalmente, permite dividir o paralelizar el trabajo.

Por ejemplo cuando tengamos que procesar una gran cantidad de datos (un fichero muy grande), podemos dividir el mismo en diez partes, y cada máquina se encargará de una décima parte del fichero, y al final lo uniremos. Con esto estamos ganando velocidad, y la velocidad es clave en el mundo del Big Data.

Existe una gran flexibilidad e interconexión con otros módulos de Apache como Hadoop, Hive o Kafka.

Posee distintas etapas pueden incluir desde soporte para análisis interactivo de datos con SQL a la creación de complejos pipelines de Machine Learning y procesamiento en streaming, todo usando el mismo motor de procesamiento y las mismas APIs.

Algunas de las evoluciones que supone Spark frente a su predecesor como Hadoop son el procesamiento en memoria que disminuye las operaciones de lectura/escritura, la posibilidad de análisis interactivo con SQL (similar a Hive) y la facilidad para interactuar con múltiples sistemas de almacenamiento persistente, los HDFS que son sistemas de procesamiento (MapReduce) de manera muy integrada.

Ecosystem

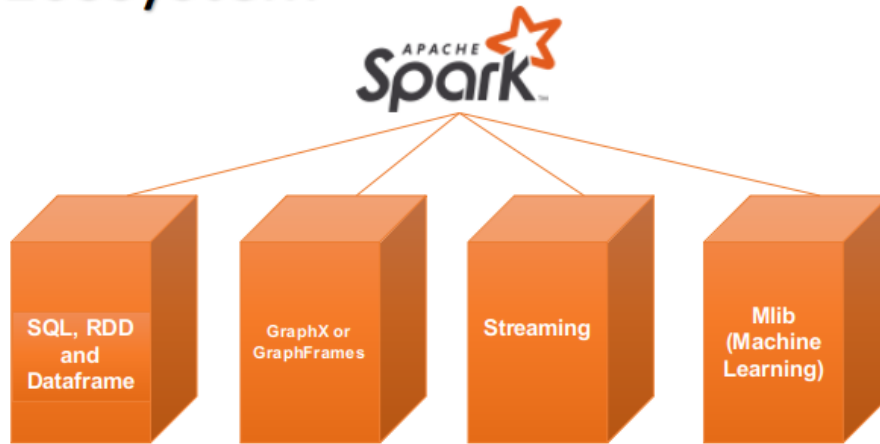


Figura 1.1 Ecosistema Apache Spark

Ecosistema Apache Spark:

| Spark RDD, SQL y dataframes

Spark RDD estructura básica de spark. Los conjuntos de datos distribuidos resistentes (RDD) es una colección distribuida inmutable de objetos. Cada conjunto de datos en RDD se divide en particiones lógicas, que se pueden calcular en diferentes nodos del clúster.

Formalmente, un RDD es una colección de registros particionada de solo lectura. Los RDD se pueden crear mediante operaciones deterministas en datos en almacenamiento estable u otros RDD. RDD es una colección de elementos tolerantes a fallas que se pueden operar en paralelo.

Spark SQL es un módulo de Spark para el procesamiento de datos estructurados. A diferencia de la API básica de Spark RDD, las interfaces proporcionadas por Spark SQL brindan a Spark más información sobre la estructura tanto de los datos como del cálculo que se está realizando. Internamente, Spark SQL usa esta información adicional para realizar optimizaciones adicionales.

Hay varias formas de interactuar con Spark SQL, incluidas SQL y la API de conjunto de datos. Al calcular un resultado, se utiliza el mismo motor de ejecución, independientemente de qué API / lenguaje esté utilizando para expresar el cálculo.

Spark DataFrame es una colección distribuida de datos organizados en columnas con nombre. Es conceptualmente equivalente a una tabla en una base de datos relacional. Los DataFrames se pueden construir a partir de una amplia gama de fuentes, como archivos de datos estructurados, tablas en Hive, bases de datos externas o RDD existentes.

| Graph X o GraphFrames

Spark GraphX es un marco de procesamiento de gráficos distribuido para proporcionar una interfaz sencilla, fácil de usar y rica para la computación de gráficos y la minería de gráficos, lo que facilita enormemente la demanda de procesamiento de gráficos distribuidos.

Combina las ventajas del paralelismo de gráficos y el paralelismo de datos. Aunque el rendimiento de un segmento informático puro no es tan bueno como GraphLab y otros marcos informáticos, si observa todo el proceso de procesamiento de gráficos (construcción de gráficos, fusión de figuras, el resultado final de la consulta), entonces el rendimiento es muy competitivo.

El procesamiento distribuido o paralelo de gráficos divide el gráfico en varios subgráficos y luego calcula estos subgráficos por separado. Al calcular, puede realizar cálculos iterativamente en etapas, es decir, realizar cálculos paralelos en el gráfico.

Spark GraphX es una expresión abstracta de una estructura de "gráfica" en el mundo real basada en la **"teoría de grafos"** y un modelo de cálculo basado en esta estructura de datos. Generalmente, en el cálculo de gráficos, la expresión de la estructura de datos básica es: $G = (V, E, D)$ donde V = vértice (vértice o nodo), E = borde (borde) y D = datos (peso).

| Streaming

Apache Spark Streaming es una extensión de la API core de Spark, que da respuesta al procesamiento de datos en tiempo real de forma escalable, con alto rendimiento y tolerancia a fallos.

| MLib (Machine Learning)

MLlib es la biblioteca de aprendizaje automático (ML) de Spark. Su objetivo es hacer que el aprendizaje automático práctico sea escalable y fácil. A alto nivel, proporciona herramientas como:

- Algoritmos ML: algoritmos de aprendizaje comunes como clasificación, regresión, agrupamiento y filtrado colaborativo
- Caracterización: extracción, transformación, reducción de dimensionalidad y selección de características
- Pipelines: herramientas para construir, evaluar y ajustar ML Pipelines
- Persistencia: guardar y cargar algoritmos, modelos y canalizaciones
- Utilidades: álgebra lineal, estadística, manejo de datos, etc.

2. INSTALACIÓN APACHE SPARK

Iremos a Docker Hub y descargaremos la imagen pyspark-notebook:

https://hub.docker.com/r/jupyter/pyspark-notebook/tags?page=1&ordering=last_updated

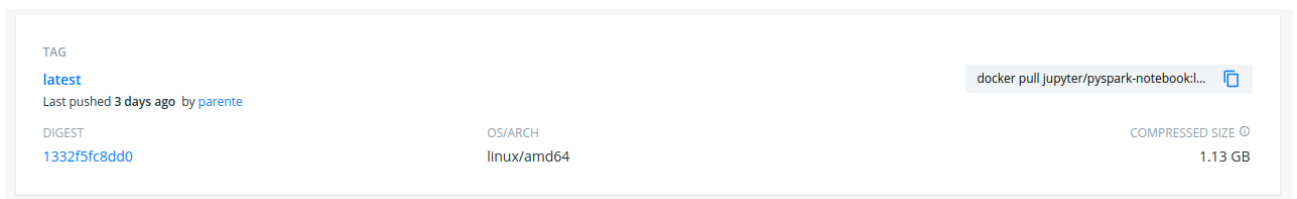


Figura 2.1 Imagen docker PySpark-notebook

Para descargarla pondremos:

```
sudo docker pull jupyter/pyspark-notebook:latest
```

```
isabel@isabel-SVE1512E1EW:~$ sudo docker pull jupyter/pyspark-notebook:latest
latest: Pulling from jupyter/pyspark-notebook
c549ccf8d472: Already exists
f2fd91df5af3: Pull complete
c7be18fa0127: Pull complete
8ad1ccf8be6e: Pull complete
27ed94f12538: Pull complete
92e3134c5187: Pull complete
8470c721240d: Pull complete
8118115d9358: Pull complete
06de358ba209: Pull complete
1d689543e03c: Pull complete
ccfb53173f0e: Pull complete
d1b342108488: Pull complete
2ae963668c1e: Pull complete
64cd025df78c: Pull complete
87161b2f4627: Pull complete
4f4fb700ef54: Pull complete
5ae87a0d7890: Pull complete
9a9deec2bc35: Pull complete
09e83b41fb06: Pull complete
b570cbf4d993: Pull complete
1234e807e4a9: Pull complete
d6a10de89b83: Pull complete
207d80c83e01: Pull complete
Digest: sha256:1332f5fc8dd03a1fb2e5fc46ed261b3dd831dd26355492bfe3dce60ff193621f
Status: Downloaded newer image for jupyter/pyspark-notebook:latest
docker.io/jupyter/pyspark-notebook:latest
isabel@isabel-SVE1512E1EW:~$
```

Figura 2.2 Descarga de la imagen docker PySpark-notebook

Para ver si la imagen esta correctamente descargada ponemos:

```
sudo docker images
```

```
isabel@isabel-SVE1512E1EW:~$ sudo docker images
REPOSITORY          TAG             IMAGE ID        CREATED         SIZE
jupyter/pyspark-notebook  latest         9ae26b715efa   2 days ago     3.17GB
postgres            latest         eeb5ef226f19   2 weeks ago    315MB
adminer             latest         203328a6b506   3 weeks ago    90MB
mongo              latest         0e120e3fce9a   5 weeks ago    449MB
isabel@isabel-SVE1512E1EW:~$
```

Figura 2.3 Imágenes Docker descargadas

Ejecutar para que guarde nuestro trabajo en -v /Users/<user>/notebooks:

```
sudo docker run -it --rm -p 8888:8888 -v /Users/isabel/notebooks:/home/jovyan/work jupyter/pyspark-notebook
```

```
isabel@isabel-SVE1512E1EW:~$ sudo docker run -it --rm -p 8888:8888 -v /Users/isabel/notebooks:/home/jovyan/work jupyter/pyspark-notebook
WARN: Jupyter Notebook deprecation notice https://github.com/jupyter/docker-stacks#jupyter-notebook-deprecation-notice.
/usr/local/bin/start-notebook.sh: running hooks in /usr/local/bin/before-notebook.d
/usr/local/bin/start-notebook.sh: running /usr/local/bin/before-notebook.d/spark-config.sh
/usr/local/bin/start-notebook.sh: done running hooks in /usr/local/bin/before-notebook.d
Executing the command: jupyter notebook
[I 17:26:38.357 NotebookApp] Writing notebook server cookie secret to /home/jovyan/.local/share/jupyter/runtime/notebook_cookie_secret
[W 2021-07-29 17:26:38.410 LabApp] 'ip' has moved from NotebookApp to ServerApp. This config will be passed to ServerApp. Be sure to update your config before our next release.
[W 2021-07-29 17:26:38.410 LabApp] 'port' has moved from NotebookApp to ServerApp. This config will be passed to ServerApp. Be sure to update your config before our next release.
[W 2021-07-29 17:26:38.468 LabApp] 'port' has moved from NotebookApp to ServerApp. This config will be passed to ServerApp. Be sure to update your config before our next release.
[W 2021-07-29 17:26:38.479 LabApp] 'port' has moved from NotebookApp to ServerApp. This config will be passed to ServerApp. Be sure to update your config before our next release.
[I 2021-07-29 17:26:38.468 LabApp] JupyterLab extension loaded from /opt/conda/lib/python3.9/site-packages/jupyterlab
[I 2021-07-29 17:26:38.468 LabApp] JupyterLab application directory is /opt/conda/share/jupyter/lab
[I 17:26:38.479 NotebookApp] Serving notebooks from local directory: /home/jovyan
[I 17:26:38.479 NotebookApp] Jupyter Notebook 6.4.0 is running at:
[I 17:26:38.479 NotebookApp] http://b9319dc69b17:8888/?token=438f38f78c499177c91709083f1504adc17381a0a3dcaaa5
[I 17:26:38.479 NotebookApp] or http://127.0.0.1:8888/?token=438f38f78c499177c91709083f1504adc17381a0a3dcaaa5
[I 17:26:38.479 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
[C 17:26:38.487 NotebookApp]

To access the notebook, open this file in a browser:
file:///home/jovyan/.local/share/jupyter/runtime/nbserver-7-open.html
Or copy and paste one of these URLs:
http://b9319dc69b17:8888/?token=438f38f78c499177c91709083f1504adc17381a0a3dcaaa5
or http://127.0.0.1:8888/?token=438f38f78c499177c91709083f1504adc17381a0a3dcaaa5
```

Figura 2.4 Ejecución de la imagen pyspark

Copiamos la ruta con el token:

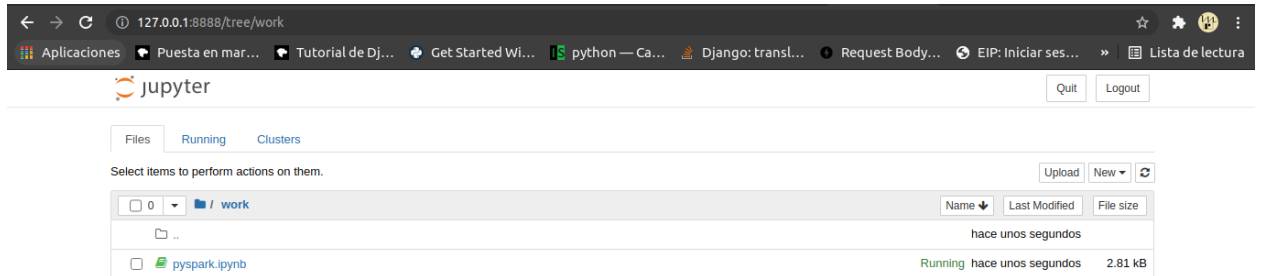


Figura 2.5 Navegador donde se ve la ejecución de pyspark-notebook

Si vamos a nuestro ordenador en la ruta `/Users/<user>/notebook` podremos los archivos que vamos a usar o crear en nuestro jupyter notebook, deberemos dar los permisos a la carpeta para que podamos crear archivos en ella:

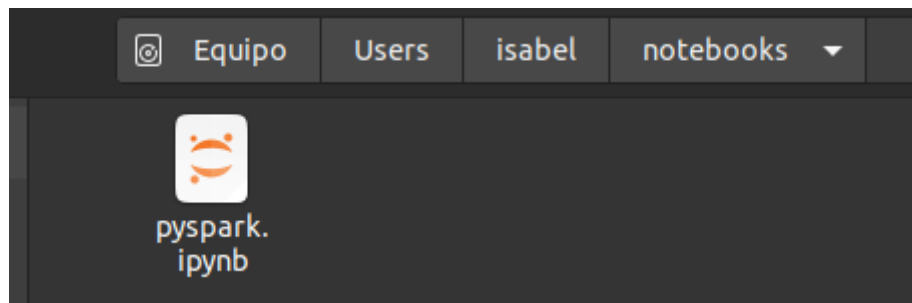


Figura 2.6 Ruta `/Users/<user>/notebooks` donde daremos permisos escritura y lectura.

```
sudo chmod 777 /Users/<user>/notebooks
```

```
isabel@isabel-SVE1512E1EW:/$ sudo chmod 777 /Users/isabel/notebooks
isabel@isabel-SVE1512E1EW:/$
```

Figura 2.7 Ruta `/Users/<user>/notebooks` donde daremos permisos escritura y lectura.

3. SPARK SQL Y DATAFRAME

Descargamos el dataset de la página (<https://data.vermont.gov/Finance/Vermont-Vendor-Payments/786x-sbp3>) y damos upload donde cargamos el csv al entorno:



Figura 3.1 Cargar el dataset en la carpeta work del jupyter

Creamos un archivo pyspark donde ejecutaremos nuestros ejemplos para cada módulo. A continuación importamos las librerías necesarias y creamos la sesión para trabajar en spark:

```
# import necessary libraries
import pandas as pd
import numpy
import matplotlib.pyplot as plt
from pyspark.sql import SparkSession

# create sparksession
spark = SparkSession \
    .builder \
    .appName("Pysparkexample") \
    .config("spark.some.config.option", "some-value") \
    .getOrCreate()
```

Cargamos el dataset:

```
df = spark.read.csv('Vermont_Vendor_Payments.csv', header='true', inferSchema
= True)
df = df.withColumn("Amount", df["Amount"].cast("double"))
```

```
In [2]: # import necessary libraries
import pandas as pd
import numpy
import matplotlib.pyplot as plt
from pyspark.sql import SparkSession
# create sparksession
spark = SparkSession \
    .builder \
    .appName("Pysparkexample") \
    .config("spark.some.config.option", "some-value") \
    .getOrCreate()

In [7]: df = spark.read.csv('Vermont Vendor Payments.csv', header='true', inferSchema = True)
df = df.withColumn("Amount", df["Amount"].cast("double"))
```

Figura 3.2 Crear la sesión para trabajar con Spark

Podemos ver los nombres de columnas mediante:

```
#we can use the columns attribute just like with pandas
columns = df.columns
print('The column Names are:')
for i in columns:
    print(i)
```

```
In [8]: #we can use the columns attribute just like with pandas
columns = df.columns
print('The column Names are:')
for i in columns:
    print(i)

The column Names are:
Quarter Ending
Department
UnitNo
Vendor Number
Vendor
City
State
DeptID Description
DeptID
Amount
Account
AcctNo
Fund Description
Fund
```

Figura 3.3 Salida del nombre de las columnas

Mostrar el número de filas en relación al número de columnas:

```
print('The total number of rows is:', df.count(), '\nThe total number of
columns is:', len(df.columns))

In [9]: print('The total number of rows is:', df.count(), '\nThe total number of columns is:', len(df.columns))

[Stage 10:> (0 + 4) / 4]

The total number of rows is: 1714538
The total number of columns is: 14
```

Figura 3.4 Salida del número filas y columnas

Podemos observar que tiene casi 2 millones de filas por 14 columnas.

Para mostrar las primeras 5 filas:

```
#show first 5 rows
df.show(5)
```

```
In [10]: #show first 5 rows
df.show(5)
```

Quarter Ending	DeptID	Amount	Department	UnitNo	Vendor Number	Vendor	City	State	DeptID	Description
12/31/2019	9150	0000002188	Vermont Housing & Cons...	9150	0000002188	Vermont Housing & Cons...	Montpelier	VT		Trust
12/31/2019	9150	0000375660	Transfer Out - Co...	720010	0000375660	Housing & Conserv...	Brattleboro	VT		VT RE
12/31/2019	9150	0000043371	Other Direct Gran...	552990	0000043371	Housing & Conserv...	Montpelier	VT		Trust
12/31/2019	9150	0000042844	Other Direct Gran...	552990	0000042844	Housing & Conserv...	Burlington	VT		Farm Viability-VH
12/31/2019	9150	0000160536	Other Direct Gran...	552990	0000160536	Housing & Conserv...	Montpelier	VT		Farm Viability-VH

only showing top 5 rows

Figura 3.5 Mostrar las primeras 5 filas del dataframe

Para visualizar la primera fila:

```
#show first row
df.head()
```

```
In [11]: #show first row
df.head()
```

```
Out[11]: Row(Quarter Ending='12/31/2019', Department='Vt Housing & Conserv Board', UnitNo=9150, Vendor Number='0000002188', Vendor='Vermont Housing & Conservation Board', City='Montpelier', State='VT', DeptID Description='Trust', DeptID='9150120000', Amount=1075000.0, Account='Transfer Out - Component Units', AcctNo='720010', Fund Description='Housing & Conserv Trust Fund', Fund='90610')
```

Figura 3.6 Mostrar la primera fila del dataframe

Descripción del dataset:

```
df.describe().show()
```

```
In [12]: df.describe().show()
```

[Stage 14:=====> (3 + 1) / 4]

	Quarter Ending	Department	UnitNo	Vendor Number	Vendor	City
State DeptID	Description	DeptID	Amount	Account	AcctNo	Fund
count	1714538	1714538	1714538	1714538	1714538	972215
1714490	1714001	1714538	1714187	1714538	1714538	
1714536	1714537					
mean	null	null	4066.099494441068	105899.06434975739	null	0.0
5151515151515151		null	4.0674150891768756E9	185136.9153755308	7.047635113583219E8	532221.4964487
54	517499.7797356828	25998.45324564796				
stddev	null	null	2330.9352198984225	121984.80012937963	null	0.0
0.605508422766933		null	2.330581000053306E9	1.4150774880903943E7	5.672550213285482E8	30184.612746648
356	4461.381794650694	19269.435003621835				
min	03/31/2010	AOT Proprietary F...	1100	0000000002	Jewett,Martin A ...	0
0	Admin.	CCV	-2880183.34	-294.00	-294.00	
507200	10000					
max	12/31/2019	Women's Commission	9150	SINGLE	xAd, Inc. w Berlin	
ZZ	Youth at Risk	Seg	6.10001E9	Youth Development...	Water/Sewer	Youth Subst
ance A...	Facilities Operat...					

Figura 3.7 Resumen del dataframe

Se usan mediante instrucciones de SQL como seleccionar las columnas en la tabla VermontVendor y mostrar los primeros 10 valores:

```
df.createOrReplaceTempView('VermontVendor')
spark.sql(
'''
SELECT `Quarter Ending`, Department, Amount, State FROM VermontVendor
LIMIT 10
'''
).show()
```

```
In [13]: # I will start by creating a temporary table query with SQL
df.createOrReplaceTempView('VermontVendor')
spark.sql(
'''
SELECT `Quarter Ending`, Department, Amount, State FROM VermontVendor
LIMIT 10
'''
).show()
```

Quarter Ending	Department	Amount	State
12/31/2019	Vt Housing & Cons...	1075000.0	VT
12/31/2019	Vt Housing & Cons...	4612.5	VT
12/31/2019	Vt Housing & Cons...	112916.67	VT
12/31/2019	Vt Housing & Cons...	17152.74	VT
12/31/2019	Vt Housing & Cons...	4850.0	VT
12/31/2019	Vt Housing & Cons...	1755.0	VT
12/31/2019	Vt Housing & Cons...	26837.54	VT
12/31/2019	Vt Housing & Cons...	30396.35	VT
12/31/2019	Vt Housing & Cons...	5430.17	VT
12/31/2019	Vt Housing & Cons...	1000.0	VT

Figura 3.8 Selección de datos

Es igual que poner en formato de dataframe:

```
df.select('Quarter Ending', 'Department', 'Amount', 'State').show(10)
```

```
In [14]: df.select('Quarter Ending', 'Department', 'Amount', 'State').show(10)
```

Quarter Ending	Department	Amount	State
12/31/2019	Vt Housing & Cons...	1075000.0	VT
12/31/2019	Vt Housing & Cons...	4612.5	VT
12/31/2019	Vt Housing & Cons...	112916.67	VT
12/31/2019	Vt Housing & Cons...	17152.74	VT
12/31/2019	Vt Housing & Cons...	4850.0	VT
12/31/2019	Vt Housing & Cons...	1755.0	VT
12/31/2019	Vt Housing & Cons...	26837.54	VT
12/31/2019	Vt Housing & Cons...	30396.35	VT
12/31/2019	Vt Housing & Cons...	5430.17	VT
12/31/2019	Vt Housing & Cons...	1000.0	VT

only showing top 10 rows

Figura 3.9 Selección de datos

Filtrar porque el departamento sea de educación:

```
spark.sql(
    '''SELECT `Quarter Ending`, Department, Amount, State FROM VermontVendor
    WHERE Department = 'Education'
    LIMIT 10'''
).show()
```

```
In [15]: spark.sql(
    ...
    SELECT `Quarter Ending`, Department, Amount, State FROM VermontVendor
    WHERE Department = 'Education'
    LIMIT 10
    ...
    ).show()
[Stage 19:=====> (1 + 3) / 4]
```

Quarter Ending	Department	Amount	State
12/31/2012	Education	302.12	VT
12/31/2012	Education	531548.0	VT
12/31/2012	Education	14082.0	VT
12/31/2012	Education	5337.66	VT
12/31/2012	Education	164436.0	VT
12/31/2012	Education	8295.0	VT
12/31/2012	Education	646.5	VT
12/31/2012	Education	29.9	VT
12/31/2012	Education	34159.0	VT
12/31/2012	Education	2626.0	VT

Figura 3.10 Selección de datos

Lo mismo que:

```
df.select('Quarter Ending', 'Department', 'Amount', 'State').filter(df['Department'] == 'Education').show(10)
```

```
In [16]: df.select('Quarter Ending', 'Department', 'Amount', 'State').filter(df['Department'] == 'Education').show(10)
```

Quarter Ending	Department	Amount	State
12/31/2012	Education	302.12	VT
12/31/2012	Education	531548.0	VT
12/31/2012	Education	14082.0	VT
12/31/2012	Education	5337.66	VT
12/31/2012	Education	164436.0	VT
12/31/2012	Education	8295.0	VT
12/31/2012	Education	646.5	VT
12/31/2012	Education	29.9	VT
12/31/2012	Education	34159.0	VT
12/31/2012	Education	2626.0	VT

only showing top 10 rows

Figura 3.11 Selección de datos

Gráfico de estos datos:

```
plot_df = spark.sql(
    '''SELECT Department, SUM(Amount) as Total FROM VermontVendor
    GROUP BY Department
    ORDER BY Total DESC
    LIMIT 10'''
).toPandas()
```

```
fig,ax = plt.subplots(1,1,figsize=(10,6))
plot_df.plot(x = 'Department', y = 'Total', kind = 'barh', color = 'C0', ax =
ax, legend = False)
ax.set_xlabel('Department', size = 16)
ax.set_ylabel('Total', size = 16)
plt.savefig('barplot.png')
plt.show()
```

```
In [17]: plot_df = spark.sql(
    '''
    SELECT Department, SUM(Amount) as Total FROM VermontVendor
    GROUP BY Department
    ORDER BY Total DESC
    LIMIT 10
    ''').toPandas()

fig,ax = plt.subplots(1,1,figsize=(10,6))
plot_df.plot(x = 'Department', y = 'Total', kind = 'barh', color = 'C0', ax = ax, legend = False)
ax.set_xlabel('Department', size = 16)
ax.set_ylabel('Total', size = 16)
plt.savefig('barplot.png')
plt.show()
```

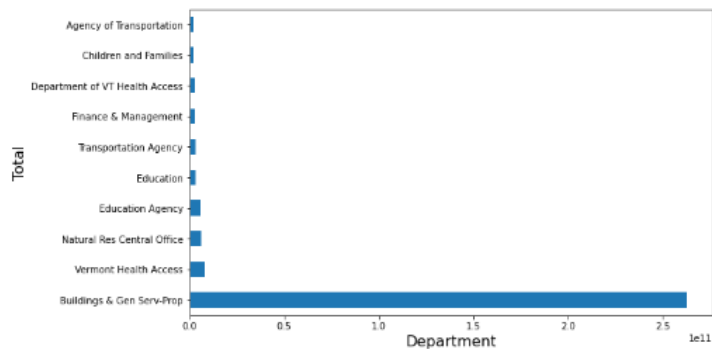


Figura 3.12 Gráficos

Usando seaborn:

```
import numpy as np
import seaborn as sns
```

```
plot_df2 = spark.sql(
'''
SELECT Department, SUM(Amount) as Total FROM VermontVendor
GROUP BY Department
''')
).toPandas()
```

```
plt.figure(figsize = (10,6))
sns.distplot(np.log(plot_df2['Total']))
plt.title('Histogram of Log Totals for all Departments in Dataset', size = 16)
plt.ylabel('Density', size = 16)
plt.xlabel('Log Total', size = 16)
plt.savefig('distplot.png')
plt.show()
```

```
In [18]: import numpy as np
import seaborn as sns
plot_df2 = spark.sql(
'''
SELECT Department, SUM(Amount) as Total FROM VermontVendor
GROUP BY Department
''')
).toPandas()
plt.figure(figsize = (10,6))
sns.distplot(np.log(plot_df2['Total']))
plt.title('Histogram of Log Totals for all Departments in Dataset', size = 16)
plt.ylabel('Density', size = 16)
plt.xlabel('Log Total', size = 16)
plt.savefig('distplot.png')
plt.show()

/opt/conda/lib/python3.9/site-packages/seaborn/distributions.py:2557: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
warnings.warn(msg, FutureWarning)
```

Figura 3.13 Gráficos

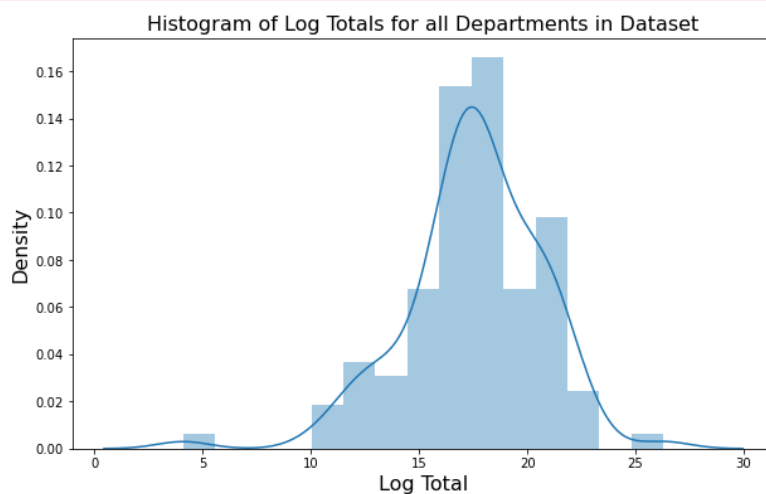


Figura 3.14 Gráficos

4. SPARK RDD

Pyspark RDD (Resilient Distributed Dataset) es una colección de objetos similar a la lista en Python. Beneficios: baja memoria del proceso, inmutable, tolerancia a fallos, partición, evolución perezosa.

Se crea una `sparkContext` que necesita declararse el máster que corresponde al clúster, pondremos `local[x]` corre en modo Standalone, donde `x` es un número entero superior a 0, que es el número de particiones que debe usar el RDD, dataframe y dataset. `x` debe ser el número de núcleos CPU que tienes.

Para ello crearemos el contexto y crearemos una lista con 1000 datos:

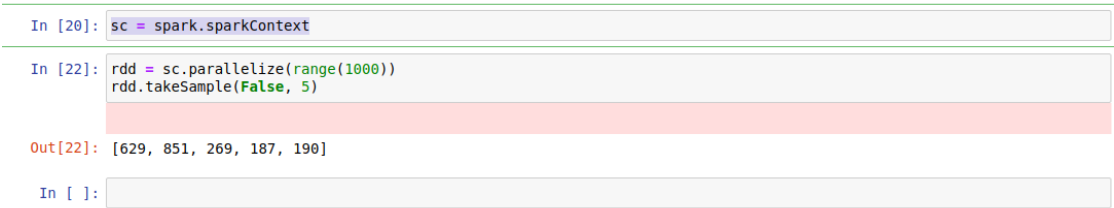
```
sc = spark.sparkContext

# RDD

rdd = sc.parallelize(range(1000))

rdd.takeSample(False, 5)
```

RDD



```
In [20]: sc = spark.sparkContext

In [22]: rdd = sc.parallelize(range(1000))
         rdd.takeSample(False, 5)

Out[22]: [629, 851, 269, 187, 190]

In [ ]:
```

Figura 4.1 Crear el contexto y paralelizar los datos

Funciona correctamente vemos como nos muestra una lista con 5 valores.

Cuando realizamos un RDD los datos se dividen basándose en la disponibilidad de recursos, cuando se ejecuta en un portátil creará el mismo número de particiones como núcleos disponibles en el sistema, usaremos para comprobarlo:

```
print("initial partition count:"+str(rdd.getNumPartitions()))
```

```

In [23]: print("initial partition count:"+str(rdd.getNumPartitions()))
initial partition count:4

```

Figura 4.2 Ver el número de particiones que usa

Para un número de particiones distinta usamos:

```

reparRdd = rdd.repartition(2)
print("re-partition count:"+str(reparRdd.getNumPartitions()))

```

Dar un número de particiones distintos menor a 4 en mi caso:

```

In [50]: reparRdd = rdd.repartition(2)
print("re-partition count:"+str(reparRdd.getNumPartitions()))
re-partition count:2

```

Figura 4.3 Ver el número de particiones que usa

Mostrar esa lista completa:

```

rdd.collect()

```

Mostrar los datos

```

In [17]: rdd.collect()

```

```

981,
982,
983,
984,
985,
986,
987,
988,
989,
990,
991,
992,
993,
994,
995,
996,
997,
998,
999]

```

Figura 4.4 Mostrar todos los datos

Contar el número de datos:

```
rdd.count()
```

```
In [18]: rdd.count()
Out[18]: 1000

In [ ]:
```

Figura 4.5 Número de datos totales

Usar una función mediante foreach():

```
def f(x): print(x)
fore = rdd.foreach(f)
```

```
In [19]: def f(x): print(x)
fore = rdd.foreach(f)

730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749

In [ ]:
```

Figura 4.6 Mostrar los valores mediante una función

Filtrar datos:

```
rdd_filter = rdd.filter(lambda x: x == 30)
filtered = rdd_filter.collect()
filtered
```

```
In [24]: rdd_filter = rdd.filter(lambda x: x == 30)
filtered = rdd_filter.collect()
filtered

Out[24]: [30]

In [ ]:
```

Figura 4.7 Filtrar datos

Primer valor del rdd:

```
# Action - first
firstRec = rdd.first()
print("First Record : "+str(firstRec))
```

Primer valor del RDD:

```
In [32]: # Action - first
firstRec = rdd.first()
print("First Record : "+str(firstRec))
First Record : 0
```

Figura 4.8 Mostrar el primer valor del RDD

Saber el valor más alto:

```
# Action - max
datMax = rdd.max()
print("Max Record : "+str(datMax))
```

Primer valor del RDD:

Valor mas alto:

```
In [33]: # Action - max
datMax = rdd.max()
print("Max Record : "+str(datMax))
Max Record : 999
```

Figura 4.9 Mostrar el máximo valor del RDD

Mostrar los tres valores primeros:

```
# Action - take
data3 = rdd.take(3)
data3
```

Mostrar 3 valores:

```
In [41]: # Action - take
data3 = rdd.take(3)
data3
Out[41]: [0, 1, 2]
```

Figura 4.10 Mostrar los 3 primeros valores del RDD

PySpark `cache()` es una clase interna `persist()` en el `sparkSession.sharedState.cacheManager.cacheQuery`:

Para ver donde se guarda la información usamos:

```
cachedRdd = rdd.cache()
```

```
cachedRdd
```

Guarda los datos automáticamente en el caché que se encuentra en:

```
In [42]: cachedRdd = rdd.cache()
cachedRdd
Out[42]: PythonRDD[84] at collect at /tmp/ipykernel_33/1896220800.py:1
```

Figura 4.11 Mostrar donde se guarda la información del RDD

Almacenar en memoria:

```
import pyspark
```

```
dfPersist = rdd.persist(pyspark.StorageLevel.MEMORY_ONLY)
```

```
dfPersist
```

Almacenar sólo en memoria:

```
In [47]: import pyspark
dfPersist = rdd.persist(pyspark.StorageLevel.MEMORY_ONLY)
dfPersist
Out[47]: PythonRDD[84] at collect at /tmp/ipykernel_33/1896220800.py:1
```

Figura 4.12 Mostrar donde se guarda la información del RDD

Para dejar de guardar en memoria:

```
rddPersist2 = dfPersist.unpersist()
```

```
rddPersist2.is_cached
```

No guardar en memoria:

```
In [54]: rddPersist2 = dfPersist.unpersist()
rddPersist2.is_cached
Out[54]: False
```

Figura 4.13 Dejar de guardar en memoria

Hay varios tipos de almacenamiento que se añadirá a `pyspark.StorageLevel` son:

- | **MEMORY_ONLY:** esto es por defecto el comportamiento de RDD lo guarda en `cache()` y almacena el RDD como objetos deserializados en la memoria JVM. Cuando no hay suficiente memoria disponible, no se guardará en RDD de algunas particiones y estas se volverán a calcular cuando sea necesario. Esto requiere más almacenamiento, pero se ejecuta más rápido, ya que se necesitan pocos ciclos de CPU para leer de la memoria.
- | **MEMORY_ONLY_SER:** Es lo mismo que `MEMORY_ONLY`, pero la diferencia es que almacena RDD como objetos serializados en la memoria JVM. Se necesita menos memoria (uso eficiente del espacio) que `MEMORY_ONLY`, ya que guarda los objetos como serializados y requiere algunos ciclos adicionales de CPU para deserializar.
- | **MEMORY_ONLY_2:** Igual que el nivel de almacenamiento `MEMORY_ONLY` pero replica cada partición en dos nodos de clúster.
- | **MEMORY_ONLY_SER_2:** Igual que el nivel de almacenamiento `MEMORY_ONLY_SER` pero replica cada partición en dos nodos de clúster.
- | **MEMORY_AND_DISK:** En este nivel de almacenamiento, el RDD se almacenará en la memoria JVM como objetos deserializados. Cuando el almacenamiento requerido es mayor que la memoria disponible, almacena algunas de las particiones sobrantes en el disco y lee los datos del disco cuando es necesario. Es más lento porque hay E / S involucradas.

MEMORY_AND_DISK_SER: Esto es lo mismo que la diferencia de nivel de almacenamiento MEMORY_AND_DISK ya que serializa los objetos RDD en la memoria y en el disco cuando no hay espacio disponible.

MEMORY_AND_DISK_2: Igual que el nivel de almacenamiento MEMORY_AND_DISK pero replica cada partición en dos nodos de clúster.

MEMORY_AND_DISK_SER_2: Igual que el nivel de almacenamiento MEMORY_AND_DISK_SER pero replica cada partición en dos nodos de clúster.

DISK_ONLY: En este nivel de almacenamiento, RDD se almacena solo en el disco y el tiempo de cálculo de la CPU es alto en función de las E / S involucradas.

DISK_ONLY_2: Igual que el nivel de almacenamiento DISK_ONLY pero replica cada partición en dos nodos de clúster.

Niveles de persistencia (definidos en `pyspark.StorageLevel`)

Nivel	Espacio	CPU	Memoria/Disco	Descripción
MEMORY_ONLY	Alto	Bajo	Memoria	Guarda el RDD como un objeto Java no serializado en la JVM. Si el RDD no cabe en memoria, algunas particiones no se <i>cachearán</i> y serán recomputadas "al vuelo" cada vez que se necesiten. Nivel por defecto en Java y Scala.
MEMORY_ONLY_SER	Bajo	Alto	Memoria	Guarda el RDD como un objeto Java serializado (un <i>byte array</i> por partición). Nivel por defecto en Python, usando <i>pickle</i> .
MEMORY_AND_DISK	Alto	Medio	Ambos	Guarda el RDD como un objeto Java no serializado en la JVM. Si el RDD no cabe en memoria, las particiones que no quepan se guardan en disco y se leen del mismo cada vez que se necesiten.
MEMORY_AND_DISK_SER	Bajo	Alto	Ambos	Similar a MEMORY_AND_DISK pero usando objetos serializados.
DISK_ONLY	Bajo	Alto	Disco	Guarda las particiones del RDD solo en disco.
OFF_HEAP	Bajo	Alto	Memoria	Guarda el RDD serializado usando memoria <i>off-heap</i> (fuera del heap de la JVM) lo que puede reducir el overhead del recolector de basura.

Figura 4.14 Tipos de almacenamiento

Broadcast

Broadcast son variables compartidas de solo lectura que se almacenan en caché y están disponibles en todos los nodos de un clúster para que las tareas puedan acceder a ellas o utilizarlas. En lugar de enviar estos datos junto con cada tarea, PySpark distribuye las variables de transmisión a la máquina utilizando algoritmos de transmisión eficientes para reducir los costos de comunicación.

Uno de los mejores casos de uso de PySpark RDD Broadcast es usarlo con datos de búsqueda, por ejemplo, búsquedas de código postal, estado, país, etc.

Cuando ejecuta un trabajo de PySpark RDD que tiene las variables de difusión definidas y utilizadas, PySpark hace lo siguiente:

- | PySpark divide el trabajo en etapas que han distribuido la mezcla y las acciones se ejecutan en la etapa.
- | Las etapas posteriores también se dividen en tareas.
- | PySpark transmite los datos comunes (reutilizables) necesarios para las tareas dentro de cada etapa.
- | Los datos transmitidos se almacenan en caché en formato serializado y deserializados antes de ejecutar cada tarea.
- | La transmisión de PySpark se crea utilizando el método de transmisión (v) de la clase SparkContext. Este método toma el argumento v que desea transmitir.

Ejemplo de modificación de los datos del código edición a el nombre completo usando Broadcast:

```
import pyspark
from pyspark.sql import SparkSession

spark = SparkSession.builder.appName('SparkByExamples.com').getOrCreate()

states = {"ED1":"Edición 1", "ED2":"Edición 2", "ED3":"Edición 3"}
broadcastStates = spark.sparkContext.broadcast(states)

data = [("James","Smith","SPAIN","ED2"),
        ("Michael","Rose","BOLIVIA","ED3"),
        ("Robert","Williams","ARGENTINA","ED2"),
        ("Maria","Jones","SPAIN","ED1")
]

columns = ["firstname","lastname","country","edition"]
df = spark.createDataFrame(data = data, schema = columns)
df.printSchema()
df.show(truncate=False)

def state_convert(code):
    return broadcastStates.value[code]

result = df.rdd.map(lambda x:
(x[0],x[1],x[2],state_convert(x[3]))).toDF(columns)
result.show(truncate=False)
```

```
In [52]: import pyspark
from pyspark.sql import SparkSession

spark = SparkSession.builder.appName('SparkByExamples.com').getOrCreate()

states = {"ED1":"Edición 1", "ED2":"Edición 2", "ED3":"Edición 3"}
broadcastStates = spark.sparkContext.broadcast(states)

data = [("James", "Smith", "SPAIN", "ED2"),
        ("Michael", "Rose", "BOLIVIA", "ED3"),
        ("Robert", "Williams", "ARGENTINA", "ED2"),
        ("Maria", "Jones", "SPAIN", "ED1")]

columns = ["firstname", "lastname", "country", "edition"]
df = spark.createDataFrame(data = data, schema = columns)
df.printSchema()
df.show(truncate=False)

def state_convert(code):
    return broadcastStates.value[code]

result = df.rdd.map(lambda x: (x[0], x[1], x[2], state_convert(x[3]))).toDF(columns)
result.show(truncate=False)

root
```

Figura 4.15 Ejemplo Broadcast

Resultado:

```
root
|-- firstname: string (nullable = true)
|-- lastname: string (nullable = true)
|-- country: string (nullable = true)
|-- edition: string (nullable = true)

+-----+-----+-----+-----+
|firstname|lastname|country |edition|
+-----+-----+-----+-----+
|James   |Smith   |SPAIN   |ED2    |
|Michael |Rose    |BOLIVIA |ED3    |
|Robert  |Williams|ARGENTINA|ED2    |
|Maria   |Jones   |SPAIN   |ED1    |
+-----+-----+-----+-----+

+-----+-----+-----+-----+
|firstname|lastname|country |edition |
+-----+-----+-----+-----+
|James   |Smith   |SPAIN   |Edición 2|
|Michael |Rose    |BOLIVIA |Edición 3|
|Robert  |Williams|ARGENTINA|Edición 2|
|Maria   |Jones   |SPAIN   |Edición 1|
+-----+-----+-----+-----+
```

Figura 4.16 Ejemplo Broadcast

Como vemos hemos creado un dataframe a partir de RDD mediante toDF(<nombre de las columnas>)

5. PUNTOS CLAVES



No te olvides...

- | Spark sirve para procesamiento de datos distribuidos diseñado para ser rápido
- | Spark SQL sirve para el procesamiento de datos estructurados.
- | Spark DataFrame es una colección distribuida de datos organizados en columnas con nombre.
- | Spark RDD es una colección distribuida inmutable de objetos.

